

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

ADVANCED MACHINE LEARNING  
FINAL PROJECT

---

# Classificazione di immagini di 102 specie di fiori

---

*Autori:*

Cristian Baldi - 806830- c.baldi1@campus.unimib.it  
Fabrizio Olivadese - 820864 - f.olivadesde1@campus.unimib.it  
Simone Vitali - 807792 - s.vitali@campus.unimib.it

3 febbraio 2020



## Sommario

L’obiettivo dell’elaborato è la classificazione di immagini rappresentanti 102 diverse specie di fiori. Per raggiungere l’obiettivo sono stati confrontati diversi approcci basati sul deep learning, in particolare una rete basata convoluzionale, una basata su autoencoder, una su fine tuning e una su feature extraction. Per decidere la struttura ottimale delle singole reti, così come gli iperparametri, sono state utilizzate tecniche di AutoML e Bayesian Optimization. La rete che utilizza feature extraction a partire da una VGG16 si dimostra il modello più efficace, ottenendo performance sul test set pari al 72.7% di accuracy e 90.3% di top-5 accuracy.

## 1 Introduzione

L’obiettivo del progetto è la realizzazione di una rete neurale convoluzionale per la **classificazione di 102 specie di fiori**. A differenza di altri task di classificazione sulle immagini, che si limitano a distinguere categorie diverse di oggetti, un task come quella della classificazione delle specie di fiori si prefissa di distinguere classi specifiche all’interno di una singola categoria di oggetti. Nel caso in questione le classi da identificare sono caratterizzate da un’elevata similarità extra-classe, ad esempio due fiori che hanno petali e colore simile potrebbero appartenere a specie diverse; possiamo trovare differenze rilevanti anche in intra-classe, ad esempio due fiori della stessa specie, a seconda dell’ambiente in cui sono cresciuti o delle condizioni atmosferiche di quando la foto è stata scattata, potrebbero avere manifestazioni diverse.

Per raggiungere l’obiettivo prefissato, diversi modelli vengono proposti, addestrati ed ottimizzati con tecniche di *Automated Machine Learning* al fine di individuare gli iperparametri e la struttura delle reti ottimali. I modelli vengono poi confrontati per individuare il migliore, sia in termine di accuracy che di top-5-accuracy. In particolare vengono confrontati un modello ideato da zero, un modello basato su un autoencoder convoluzionale, un modello che usa la rete *VGG16* come *feature extractor* ed un modello basato sul *fine tuning* sempre della rete *VGG16*.

## 2 Dataset



Figura 1: Alcune immagini del dataset con la rispettiva etichetta

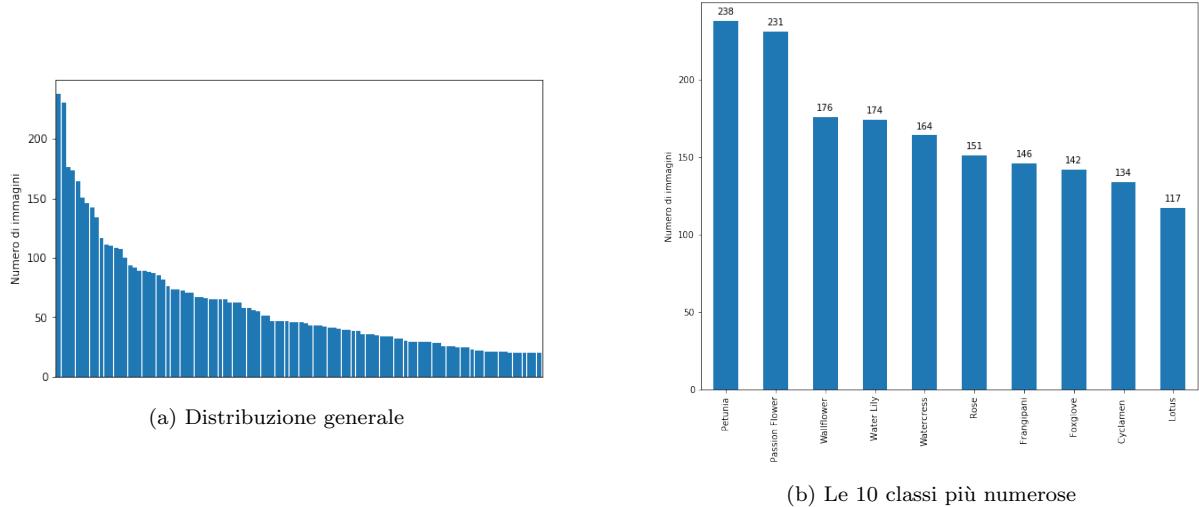


Figura 2: Distribuzione delle classi nel test set

Il dataset utilizzato per affrontare il problema è **”102 Category Flower Dataset”** presentato nel 2008 in **”Automated flower classification over a large number of classes”** [1]. Nel dataset sono presenti **8189 immagini** di diversa risoluzione, rappresentanti 102 specie diverse di fiori. A ciascuna immagine del dataset è associata un’etichetta, corrispondente alla specie del fiore rappresentato. Un esempio per ogni classe è visibile in Appendice A.

Il dataset, di cui è possibile vedere un esempio in Figura 1, è suddiviso in **1020 immagini di training** (10 immagini per specie di fiore), **1020 immagini di validation** (10 immagini per specie di fiore) e **6149 immagini di test** (distribuite secondo la Figura 2). Questa suddivisione rispecchia un caso di utilizzo reale, le immagini per l’addestramento e la validazione di un modello in genere sono poche, soprattutto quando si ha a che fare con problemi mai affrontati e di questa dimensione, mentre quelle di test, potenzialmente mai viste dagli sviluppatori del modello, possono essere molte di più, ed in genere distribuite in modo non uniforme, con preponderanza per le classi più comuni nel mondo reale.

Ogni immagine presente nel dataset rappresenta in piano un fiore e sullo sfondo altri elementi naturali o artificiali (come erba o foglie). [1] fornisce anche una versione segmentata del dataset, cioè con solo il fiore presente in primo piano e lo sfondo rimosso. E’ stato scelto di utilizzare la versione non segmentata delle immagini per verificare l’effettivo potenziale delle reti convoluzionali su immagini tanto più simili alla realtà.

Tutte le immagini sono state sottoposte ad un **preprocessing minimale**. Inizialmente tutte le immagini sono state ridimensionate per avere una risoluzione di **224px per lato**, che corrisponde inoltre alla risoluzione di input standard accettata dalla rete **VGG16**. Prima di essere utilizzate nei modelli da noi realizzati le immagini sono state riscalate per avere **valori compresi tra 0 e 1** per ogni layer e pixel. Per quanto riguarda invece i modelli di *transfer learning*, alle immagini è stata applicata la funzione di preprocessing relativa al modello applicato che, oltre a riscalarle, ne standardizza la media e la varianza secondo le caratteristiche proprie del modello.

Avendo a disposizione solo 10 esempi per ogni classe delle immagini di training, è stato necessario ricorrere a delle tecniche che permettessero di arricchire maggiormente il set proposto attraverso tecniche di ***data augmentation***. In particolare, durante la fase di training, ogni immagine, prima di essere passata alla rete neurale, può subire uno o più tra le seguenti trasformazioni: **ribaltamento orizzontale**, **ribaltamento verticale**, **zoom-in o zoom-out** nell’ordine del 20%, rotazione tra -30° e 30° e modifiche alla



Figura 3: Immagini generate con data augmentation

luminosità nell’ordine del 20%. Alcuni esempi sono mostrati nella Figura 3. Questa tecnica permette di espandere potenzialmente all’infinito il dataset di training, oltre a creare modelli più robusti che possono gestire e riconoscere immagini scattate nelle più disparate situazioni.

### 3 Approccio Metodologico

In questa sezione vengono presentati i differenti modelli di *machine learning* sviluppati per la risoluzione del problema illustrato. Di ogni modello verrà indicata: **l’intuizione** che ne sta alla base, la **struttura generale** e gli **iperparametri** relativi. Per prima cosa vengono presentati i punti in comune dei vari modelli.

Trattandosi di un problema di classificazione non-binaria la funzione di attivazione scelta per i neuroni dell’ultimo layer è la **funzione *Softmax***, scelta standard per problemi di questo genere. Per i *layer* intermedi la funzione di attivazione adottata è la ***ReLU***, sia per la sua popolarità ma anche per le sue proprietà analitiche. Come funzione di *loss* è stata scelta la ***Categorical Crossentropy***, molto comune per i problemi di classificazione non-binaria.

Sono state adottate anche **tecniche di regolarizzazione** per prevenire l’*overfitting* e favorire la generalizzazione dell’apprendimento, la principale utilizzata è quella dei ***Layer di Dropout***, tecnica che prevede l’inserimento di questi particolari *layer* in genere tra i vari *layer* densi dei classificatori. In fase di addestramento viene anche utilizzata la tecnica **dell’*Early Stopping***: ad ogni epoca viene controllato il valore ottenuto di *loss* del validation set e, se non è diminuito nelle ultime 5 epoche, l’addestramento si interrompe ripristinando i pesi del miglior modello visto fino a quel momento.

Per tutti i modelli costruiti si è scelto di utilizzare **l’ottimizzatore *Adam*** [2], perché rivelatosi più efficace e rapido nella convergenza rispetto agli altri ottimizzatori, tra cui *SGD* e *RMSProp*. Il valore di *learning rate* adottato varia da modello a modello, in particolare è stato scelto nel range 0.000001 e 0.01 da un processo di ottimizzazione.

Per la ricerca della struttura ottimale e dei valori ottimali da assegnare agli iperparametri dei modelli sono state utilizzate tecniche di ***Automated Machine Learning***. In particolare, tramite l’utilizzo della libreria *Python SMAC3* [3] è stata attuata una procedura di **ottimizzazione bayesiana** basata su un modello surrogato *Random Forest* e che utilizza ***Expected Improvement*** come funzione di acquisizione. La scelta di utilizzare proprio *Expected Improvement* come funzione di acquisizione è dettata dal fatto che a livello sperimentale si è rivelata quella più efficace per individuare la struttura ottima. Viene stabilito un budget di 30 campionamenti, a cui vanno aggiunti 6 campionamenti

di inizializzazione. Ciascun campionamento corrisponde ad un training del modello per 100 epoche (o fino ad un *early stop*, quindi al minimo di 5 epoche) ed ad una esecuzione del modello sul validation set. Come valutazione della bontà di una configurazione si è scelto il valore di *loss* calcolato sul validation set; la scelta è dettata dal fatto che si vuole ottenere un modello meno incerto, quindi con un valore di *loss* più basso possibile, piuttosto che un modello che ottiene più accuracy sul validation set ma per "casualità".

Come input per il training dei modelli vengono passati degli stream di immagini pre-processati con le tecniche di data augmentation. Questo vuol dire che, ad ogni epoca, le immagini prese in input dai modelli sono diverse tra loro, perchè ognuna viene modificata in modo diverso con la data augmentation. Questo garantisce una maggior variabilità tra i dati al fine di costruire un modello più robusto. In generale ogni modello ad ogni epoca viene addestrato su 1020 immagini sempre diverse tra le epoche e validato su altre 1020 immagini sempre uguali tra le epoche.

### 3.1 Classificatore Convoluzionale Standard

Il primo modello sviluppato consiste in una **rete deep convoluzionale fully-connected**, ideata per verificare se, anche con un modello relativamente semplice, sia possibile risolvere il problema proposto. Si è scelta una struttura convoluzionale in quanto in input al modello vengono passate delle immagini a colori.

La struttura del modello ricalca quella di altre reti convoluzionali standard più note, pochi *Kernel* di dimensione elevata nei primi *layer* più vicini all'input, e tanti *Kernel* di dimensione ridotta più ci si avvicina all'output, ottenuti tramite *layer di Max Pooling*. Nel dettaglio la rete è così strutturata:

- **Layer di Input**, per immagini RGB della dimensione di  $244 \times 244$  pixel
- **Layer Convoluzionale**, 64 Kernel  $3 \times 3$ , padding a "same" e attivazione ReLu; Output:  $224 \times 224 \times 64$
- **Layer Max Pooling**, pool size di  $2 \times 2$ ; Output:  $112 \times 112 \times 64$
- **Layer Convoluzionale**, 128 Kernel  $3 \times 3$ , padding a "same" e attivazione ReLu; Output:  $112 \times 112 \times 128$
- **Layer Max Pooling**, pool size di  $2 \times 2$ ; Dimensione output:  $56 \times 56 \times 128$
- **Layer Convoluzionale**, 256 Kernel  $3 \times 3$ , padding a "same" e attivazione ReLu; Output:  $56 \times 56 \times 256$
- **Layer Max Pooling**, pool size di  $2 \times 2$ ; Dimensione output:  $28 \times 28 \times 256$
- **Layer Convoluzionale**, 512 Kernel  $3 \times 3$ , padding a "same" e attivazione ReLu; Output:  $28 \times 28 \times 512$
- **Layer Max Pooling**, pool size di  $2 \times 2$ ; Dimensione output:  $14 \times 14 \times 512$
- **Layer Convoluzionale**, 512 Kernel  $3 \times 3$ , padding a "same" e attivazione ReLu; Output:  $14 \times 14 \times 512$
- **Layer di Flatten**, con output di 25088 valori
- Da 1 a 4 coppie di layer così strutturati:
  - **Layer di Dropout**, con tasso di dropout impostato a 0.1
  - **Layer Denso**, dalla dimensione compresa tra 32 e 4096 neuroni, con attivazione ReLu
- **Layer di Dropout**, con tasso di dropout impostato a 0.1
- **Layer Denso di Output**, della dimensione di 102 neuroni (1 per classe), con funzione di attivazione SoftMax

Il numero di coppie di layer, così come la dimensione dei layer densi viene scelta tramite AutoML.

### 3.2 Classificatore basato su Autoencoder

Per la realizzazione di questo modello è innanzitutto necessaria la creazione di un ***Autoencoder***, una particolare rete neurale che ha come obiettivo il replicare sugli output gli stessi valori (nel caso in questione immagini) che riceve in input. Un *autoencoder* in genere ha una struttura a "doppio imbuto", la dimensionalità dei dati di input viene via via diminuita da vari *layer intermedi*, quelli di *encoding*, fino a raggiungere uno strato detto "**latente**", di dimensione nettamente inferiore a quello di input. Dopo il *layer latente* la rete è composta da altri *layer di decoding*, simmetrici a quelli di *encoding*, che alla fine generano un output di dimensionalità pari a quello dell'input. Dopo un addestramento sufficiente della rete, un *autoencoder* riesce a riprodurre sugli output valori molto simili a quelli che ha in input.



Figura 4: Immagini originali (sopra) a confronto con quelle generate dall'autoencoder (sotto)

Lo strato utile per adempiere a task di classificazione è il *layer latente* prodotto dall'*autoencoder*, in quanto contiene tutte le informazioni necessarie alla ricostruzione del dato di input ma con una dimensionalità nettamente inferiore. Di conseguenza, può essere interessante realizzare un classificatore basato su una rete neurale che usa proprio il *layer latente* dell'*autoencoder* come input, con lo scopo di ridurre notevolmente la dimensionalità delle immagini originali.

In particolare l'*autoencoder* proposto per affrontare il problema è un ***autoencoder convoluzionale***, che sia come *layer intermedi* che come *layer latente* utilizza non dei semplici *layer densi* ma *layer convoluzionali*. L'intuizione alla base di questa scelta è che, date le caratteristiche delle reti convoluzionali, sarà possibile ricostruire più precisamente le immagini di input sui neuroni di output.

La struttura dell'*autoencoder*, che presenta 5,496,451 parametri addestrabili, è la seguente:

- **Layer di Input**, per immagini *RGB* della dimensione di  $244 \times 244$  pixel
- 5 coppie di layer così strutturati:
  - **Layer Convoluzionale**, [32/64/128/256/512] Kernel (a seconda della profondità del layer)  $3 \times 3$ , padding a "same" e attivazione ReLu
  - **Layer Max Pooling**, pool size di  $2 \times 2$
  - In particolare l'ultimo layer di Max Pooling, quello riferito al layer convoluzionale da 256 Kernel, è quello di rappresentazione **latente**
  - **Layer Latente**, composto da  $7 \times 7 \times 512$  parametri

- 5 coppie di layer così strutturati:
  - **Layer Convoluzionale**, [512/256/128/64/32] Kernel (a seconda della profondità del layer)  $3 \times 3$ , padding a "same" e attivazione ReLu
  - **Layer di Upsampling**, di size  $2 \times 2$
- **Layer Convoluzionale**, 3 Kernel  $3 \times 3$ , padding a "same" e attivazione Sigmoide, per la ricostruzione dell'immagine finale.

In particolare, il numero di parametri del *layer latente* è  $7 \times 7 \times 512 = 25048$ , che se confrontato con la dimensione originale dell'input ( $244 \times 244 \times 3 = 178,608$ ), permette di calcolare il tasso di compressione dell'*autoencoder* che è

$$\frac{178,608}{25,048} = 7.13$$

Per essere addestrato, l'*autoencoder* riceve in input lo stesso dataset disponibile ai vari classificatori, cioè le immagini di training a cui è stata applicata la *data augmentation*. La funzione di loss utilizzata per l'*autoencoder* è la funzione Mean Square Error, scelta abbastanza comune per gli *autoencoder*. L'*autoencoder* viene addestrato utilizzando l'*ottimizzatore Adam*, con *learning rate* pari a 0.001, per 500 epoche, o finché la *loss* dell'*autoencoder* non migliora per 5 epoche consecutive.

In seguito all'addestramento, la parte dell'*autoencoder* che si occupa dell'*encoding* (dall'input al *layer latente*) viene usata come **input di un classificatore neurale standard**. Nello specifico, l'*encoder* viene reso non più addestrabile: sono stati congelati i pesi imparati nella fase di addestramento dell'*autoencoder*, e l'*encoder* è poi stato collegato ad altri *layer* densi, ottenendo la struttura che segue:

- **Encoder convoluzionale**, output della dimensione di  $7 \times 7 \times 512$
- **Layer di Flatten**, output di  $7 \times 7 \times 512 = 25,048$  valori
- Da 1 a 4 coppie di layer così strutturati:
  - **Layer di Dropout**, con tasso di dropout impostato a 0.1
  - **Layer Denso**, dalla dimensione compresa tra 32 e 4096 neuroni, con attivazione ReLu
- **Layer di Dropout**, con tasso di dropout impostato a 0.1
- **Layer Denso di Output**, della dimensione di 102 neuroni, con funzione di attivazione SoftMax

Il numero di coppie di layer, così come la dimensione dei layer densi viene scelta tramite AutoML.

### 3.3 Classificatore Convoluzionale basato su Feature Extractor

Per questo classificatore si è scelto di utilizzare la **rete VGG16** [4] come *feature extractor* delle immagini. In particolare, la *VGG16* utilizzata è **pre-addestrata sul dataset ImageNet** [5], un task di classificazione con oltre 10,000 classi di immagini da distinguere. Il dataset usato per affrontare il problema presentato in questo documento è riconducibile a quello di *ImageNet*: *ImageNet* contiene infatti immagini da ogni tipo di realtà, tra cui anche immagini naturali, come quelle di fiori, piante, alberi; ciascuna di queste macro-classi è poi suddivisa in classi più specifiche, che vanno ad identificare la specie esatta del fiore, della pianta o dell'albero in questione. In *ImageNet*, ad esempio, è contenuta una classe per le immagini di girasoli o di rose, anche se non sono contenute tutte le

classi presenti nel dataset *Flower102*, più specifiche e limitate ad una singola realtà (i fiori). Si può quindi affermare che esiste una sovrapposizione tra le classi di *ImageNet* e quelle di *Flower102*, anche se decisamente minimale. È quindi interessante andare ad osservare come un modello addestrato su *ImageNet* possa comportarsi sul dataset preso in analisi, in quanto almeno una parte della conoscenza appresa su *ImageNet* può essere trasferibile sul dataset in analisi. Allo stesso tempo è anche sensato aspettarsi che utilizzare una rete addestrata solo su *ImageNet* possa non essere sufficiente per risolvere il task completamente.

In fase di sviluppo sono stati testati altri modelli addestrati su *ImageNet* da usare sempre come *feature extractor*, tra cui **ResNet50** ed **Inception**. Sono stati alla fine scartati in quanto le performance ottenute, anche dopo la procedura di ottimizzazione, non sono state ritenute sufficientemente adeguate rispetto a quelle ottenute con *VGG16*.

Per utilizzare la rete *VGG16* come *feature extraction* per un classificatore per prima cosa è necessario che questa venga "tagliata" ad un certo *layer* allo scopo di estrarre i valori di output delle immagini di input processati fino a quel *layer*. In genere, quando si utilizza la rete *VGG16*, data la struttura della rete, si preferisce tagliare la rete in corrispondenza dei *layer di pooling*. Ciascun *layer di pooling* infatti identifica uno stacco tra un blocco ed un altro e può essere quindi indicato come input per un nuovo classificatore. In genere per i classificatori convoluzionali, più i *layer* sono vicini all'input più le *feature* analizzate sono di alto livello (ad esempio forme, linee, ...) mentre più sono vicini all'output più i *layer* lavorano con *feature* riferite ai dettagli dell'immagine.

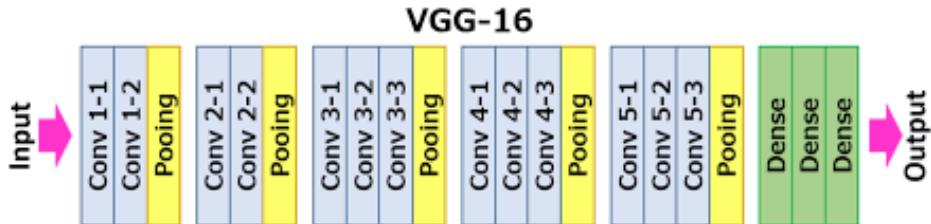


Figura 5: Composizione della rete VGG16

Scegliere fino a quale blocco della *VGG16* utilizzare per l'estrazione delle *feature* non è immediato, per questo la scelta sarà lasciata all'ottimizzazione fatta tramite *AutoML*. Una volta selezionato il blocco in cui tagliare la rete, a tutti i *layer*, dal primo fino a quel blocco, viene disabilitata la possibilità di aggiornare i propri pesi in fase di addestramento; mentre i *layer* da quel blocco in poi vengono eliminati e rimpiazzati da altri *layer* densi capaci di portare a termine la classificazione.

La struttura del classificatore che quindi si ottiene è la seguente:

- **VGG16**, con input immagini RGB dalla dimensione  $244 \times 244$  pixel. La rete è tagliata in corrispondenza di uno tra i 5 *layer* di pooling
- **Layer di Global Average Pooling**, che appiattisce il risultato del *layer* precedente tramite l'applicazione di un Average Pooling su ogni matrice convoluzionale risultata dal *layer* precedente
- Da 1 a 4 coppie di *layer* così strutturati:
  - **Layer di Dropout**, con tasso di dropout impostato a 0.1
  - **Layer Denso**, dalla dimensione compresa tra 32 e 4096 neuroni, con attivazione ReLu
- **Layer di Dropout**, con tasso di dropout impostato a 0.1

- **Layer Denso di Output**, della dimensione di 102 neuroni, con funzione di attivazione SoftMax

Il blocco convoluzionale a cui tagliare la rete VGG16, il numero di coppie di layer, così come la dimensione dei layer densi viene scelta tramite AutoML.

### 3.4 Classificatore Convoluzionale basato su Fine Tuning

Oltre che come *feature extractor*, si è scelto di valutare la performance di una rete *VGG16* addestrata su *ImageNet*, come classificatore su cui eseguire il processo di *fine tuning*. Si è deciso di realizzare un classificatore di questo tipo dato che si ha a disposizione una discreta quantità di immagini di addestramento e perciò la rete può essere perfezionata per il task del riconoscimento delle specie di fiori, anche grazie alla minima sovrapposizione di classi con *ImageNet*.

In particolare, il *fine-tuning* proposto per la soluzione nel problema rende innanzitutto alcuni *layer* della rete *VGG16* inaddestrabili (i primi  $N$  layer più vicini all'output, con  $N$  deciso tramite ottimizzazione con *AutoML*), mentre lascia gli altri liberi di cambiare i propri pesi durante l'addestramento. Inoltre, viene rimpiazzato completamente il classificatore denso posizionato dopo l'ultimo blocco convoluzionale con un classificatore denso realizzato *da zero*. La nuova struttura che si ottiene è la seguente:

- **VGG16**, fino all'ultimo blocco convoluzionale, con gli ultimi  $N$  layer addestrabili (con  $N$  che varia da 0 a 19) e con input immagini RGB dalla dimensione  $244 \times 244$  pixel; Output:  $7 \times 7 \times 512$
- **Layer di Flatten**,
- Da 1 a 4 coppie di layer così strutturati:
  - **Layer di Dropout**, con tasso di dropout impostato a 0.1
  - **Layer Denso**, dalla dimensione compresa tra 32 e 4096 neuroni, con attivazione ReLu
- **Layer di Dropout**, con tasso di dropout impostato a 0.1
- **Layer Denso di Output**, della dimensione di 102 neuroni, con funzione di attivazione SoftMax

Il valore di  $N$  per la VGG16, il numero di coppie di layer, così come la dimensione dei layer densi viene scelta tramite AutoML.

## 4 Risultati e Valutazione

In questo capitolo vengono riportati e commentati i risultati ottenuti dai vari modelli presentati. Per prima cosa, per ogni modello, viene mostrata la struttura ottimale ottenuta tramite la procedura di ottimizzazione mediante *AutoML* con il relativo grafico del miglioramento progressivo sulla *loss function* nella ricerca degli iperparametri.

Vengono poi presentate e messe a confronto le performance del modello ottimo di ogni categoria sul validation set e sul test set, prestando particolare attenzione all'accuracy e alla top-5-accuracy, computabile grazie all'output probabilistico dell'ultimo layer delle reti presentati.

Infine, per quanto riguarda il modello che ha ottenuto i riusati migliori in assoluto viene presentato un report dettagliato sulla classificazione con annessi esempi visivi di classificazione corretti e scorretti ottenuti sul training set.

## 4.1 Classificatore Convoluzionale Standard

La configurazione ottimale individuata dalla procedura di ottimizzazione è la seguente:

- N° Layer Densi: 3
  - Dimensione 1° Layer: 395
  - Dimensione 2° Layer: 74
  - Dimensione 3° Layer: 414
- Learnin Rate Adam: 0.000306

Il modello ottimizzato ha **13,923,623 parametri** addestrabili.

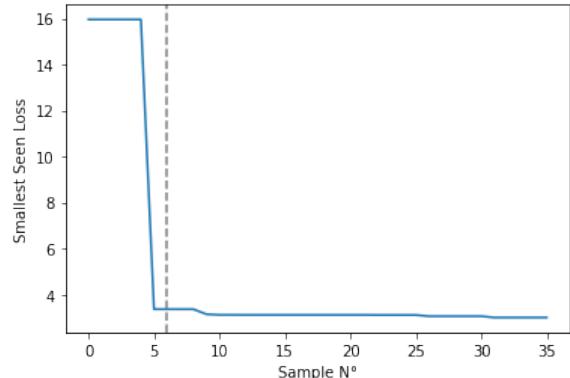


Figura 6: Migliore loss vista durante l'ottimizzazione

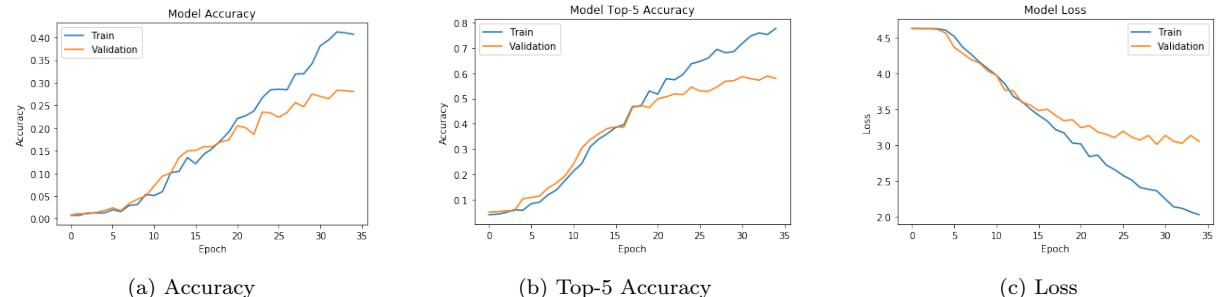


Figura 7: Performance durante l'addestramento del modello ottimizzato

Tabella 1: Performance del modello con parametri ottimali

	Accuracy	Top-5 Accuracy	Loss
<b>Training</b>	0.407	0.7772	2.024
<b>Validation</b>	0.280	0.578	3.048
<b>Testing</b>	0.221	0.502	3.337

## 4.2 Classificatore Convoluzionale basato su Autoencoder

### Autoencoder

Sono riportati alcuni dettagli sull'addestramento dell'*autoencoder* convoluzionale:

- Ottimizzatore: Adam - LR: 0.0001
- Epoche prima di Early Stop: 104
- Loss su training set (MSE): 0.0086

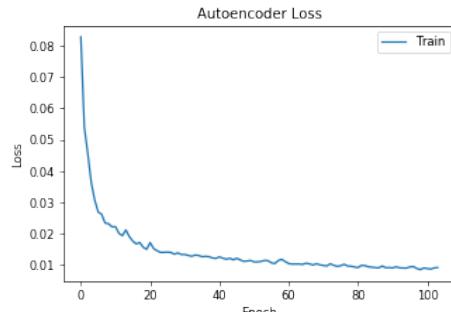


Figura 8: MSE durante l'addestramento dell'autoencoder

## Classificatore

La configurazione ottimale individuata dalla procedura di ottimizzazione è la seguente:

- N° Layer Densi: 1
  - Dimensione 1° Layer: 1138
- Learnin Rate Adam: 0.00101

Il modello ottimizzato ha **30,236,036 parametri** di cui 28,667,460 addestrabili.

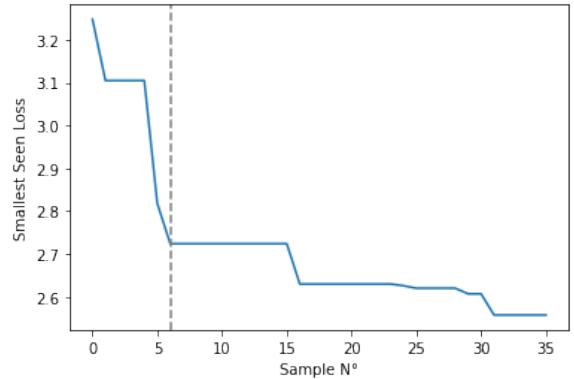


Figura 9: Migliore loss vista durante l'ottimizzazione

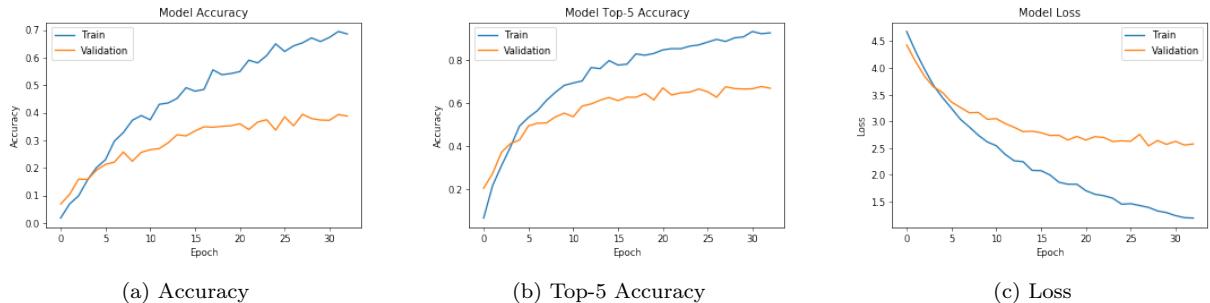


Figura 10: Performance durante l'addestramento del modello ottimizzato

Tabella 2: Performance del modello con parametri ottimali

	Accuracy	Top-5 Accuracy	Loss
<b>Training</b>	0.686	0.929	1.192
<b>Validation</b>	0.388	0.670	2.578
<b>Testing</b>	0.331	0.619	2.889

## 4.3 Classificatore Convoluzionale basato su Feature Extractor

La configurazione ottimale individuata dalla procedura di ottimizzazione è la seguente:

- Blocco convoluzionale VGG16: 5°
- N° Layer Densi: 3
  - Dimensione 1° Layer: 1849
  - Dimensione 2° Layer: 3224
  - Dimensione 3° Layer: 1994
- Learnin Rate Adam: 0.0000406

Il modello ottimizzato ha **28,261,765 parametri** di cui 13,547,077 addestrabili.

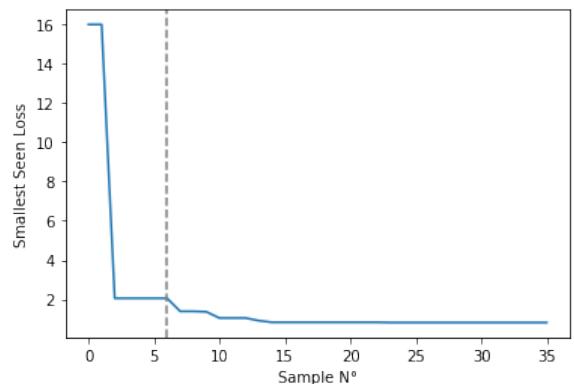


Figura 11: Migliore loss vista durante l'ottimizzazione

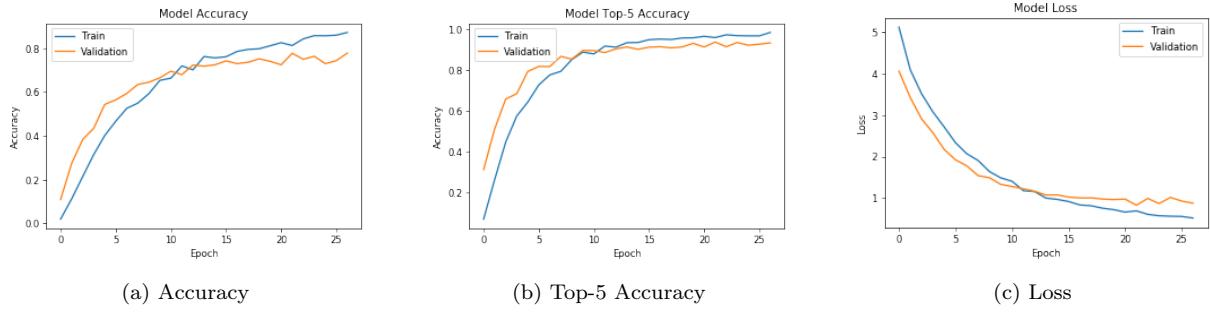


Figura 12: Performance durante l'addestramento del modello ottimizzato

Tabella 3: Performance del modello con parametri ottimali

	Accuracy	Top-5 Accuracy	Loss
<b>Training</b>	0.871	0.985	0.512
<b>Validation</b>	0.777	0.934	0.877
<b>Testing</b>	0.727	0.903	1.088

#### 4.4 Classificatore Convoluzionale basato su Fine Tuning

La configurazione ottimale individuata dalla procedura di ottimizzazione è la seguente:

- N° ultimi layer VGG16 resi addestrabili: 2
- N° Layer Densi: 4
  - Dimensione 1° Layer: 520
  - Dimensione 2° Layer: 1753
  - Dimensione 3° Layer: 54
  - Dimensione 3° Layer: 1344
- Learnin Rate Adam: 0.0000679

Il modello ottimizzato ha **28,980,107** parametri di cui 16,625,227 addestrabili.

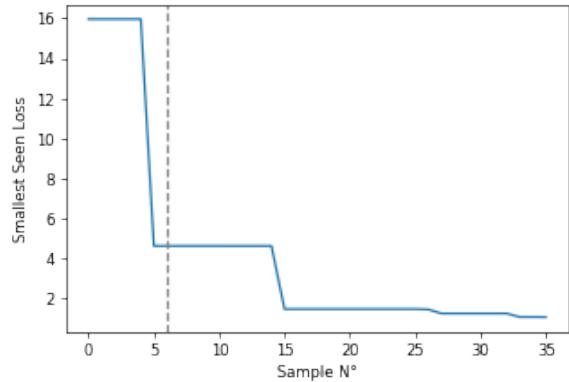


Figura 13: Migliore loss vista durante l'ottimizzazione

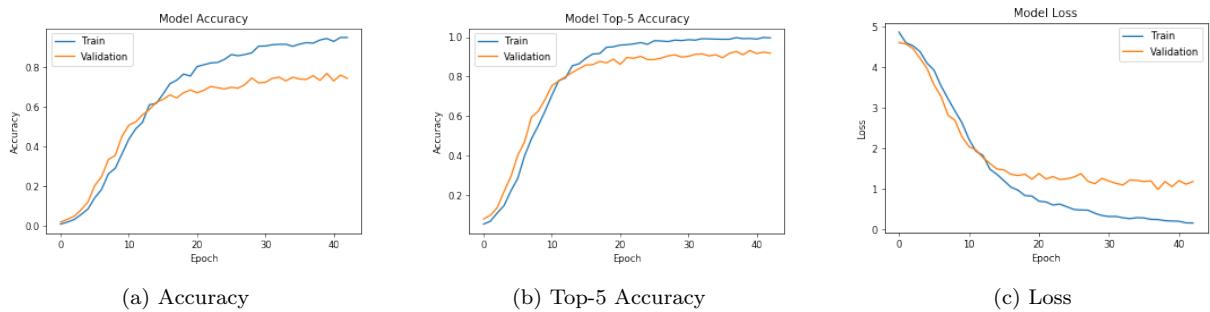


Figura 14: Performance durante l'addestramento del modello ottimizzato

Tabella 4: Performance del modello con parametri ottimali

	<b>Accuracy</b>	<b>Top-5 Accuracy</b>	<b>Loss</b>
<b>Training</b>	0.949	0.996	0.166
<b>Validation</b>	0.743	0.918	1.184
<b>Testing</b>	0.727	0.901	1.245

## 4.5 Risultati dettagliati del modello migliore

Sia il modello basato sulla *feature extraction* sia quello basato su *fine tuning* ottengono performance similari per quanto riguarda l'accuracy e la top-5 accuracy. Il modello basato su ***feature extraction*** si dimostra migliore in termine di ***loss***, di conseguenza è stato scelto come **modello migliore**.

Innanzitutto vengono riportate altre metriche medie utili alla valutazione del modello:

- Precision: 0.756/0.764 (pesata)
- Recall: 0.734/0.726 (pesata)
- F1 Score: 0.727/0.724 (pesata)

Nell'appendice B sono riportati i risultati dettagliati per ogni classe sulle metriche sopra riportate.

Sono presentati in seguito alcuni esempi di classificazione e per alcune classi i principali errori commessi in fase di testing, al fine di poter valutare visivamente la tipologia di errori commessi dal modello.



Figura 15: Esempi di classificazione



Figura 16: Principali errori di predizione sulla classe "Artichoke"



Figura 17: Principali errori di predizione sulla classe "Passion Flower"



Figura 18: Principali errori di predizione sulla classe "Lotus"

## 5 Discussione dei risultati

### Ottimizzazione dei Modelli

Per quanto riguarda l’ottimizzazione dei modelli, si può osservare, grazie ai grafici che mostrano il miglioramento delle *loss* dei modelli durante la ricerca degli iperparametri e della struttura ottimale, come la procedura sia efficace nel trovare le combinazioni ottimali di parametri. In generale **meno di 20 campionamenti** sullo spazio di ricerca sono necessari per ottenere risultati soddisfacenti, anche se, effettuando tutti e 35 i campionamenti stabili, la *loss* sul validation set continua a diminuire.

Per la maggior parte dei modelli ottimizzati, tranne che per quello basato su autoencoder, viene preferito un classificatore denso a più livelli, 3 o 4 a seconda del modello. Per tutti i modelli il learning rate per l’ottimizzatore Adam è molto basso non andando mai oltre lo 0.001, il learning rate di default dell’ottimizzatore. Le reti ottenute hanno sempre un numero elevato di parametri totali e addestrabili, la più piccola è quella basata su classificatore convoluzionale standard, anche a causa della dimensione inferiore dei layer densi di classificazione.

Sicuramente interessante da riportare è il risultato ottenuto per l’ottimizzazione del modello basato sul *fine tuning*: la procedura di ottimizzazione sceglie di rendere addestrabili solo gli **ultimi due layer** della *VGG16*, mentre tutti i *layer* precedenti vengono resi inaddestrabili. L’ultimo *layer* della *VGG16* è un *layer di pooling*, che, di conseguenza, non ha parametri addestrabili, mentre il penultimo è un *layer convoluzionale* addestrabile. In genere da una procedura di *fine tuning* ci si aspetta che più *layer* della rete su cui effettuare il *tuning* dei paramenti siano addestrabili, mentre in questo caso solo uno viene modificato, questa scelta perché conduce ad una *loss* migliore. Rendere più *layer* addestrabili avrebbe quindi peggiorato le performance sul validation set.

Per esperimenti futuri potrebbe essere interessante andare a valutare altri ottimizzatori della funzione di *loss*, al fine di verificare se le performance ottenute siano state limitate dall’uso dal solo ottimizzatore *Adam*.

### Classificazione

I risultati ottenuti sulla classificazione da alcuni dei modelli proposti sono incoraggianti, seppur con largo margine di miglioramento.

Come è semplice prevedere, il modello che si è rivelato essere il peggiore è quello **convoluzionale semplice** che nel task di classificazione sul test set ottiene un **1accuracy del 22%** e una **top-5 accuracy del 50%** (Tabella 1), dopo una procedura di addestramento di 35 epoch interrotta da *early stop* (Figura 7). Delle Performance migliori si potrebbero ottenere andando a modificare la struttura convoluzionale della rete, oltre che andando ad aumentare i dati in input: un modello appositamente creato e ottimizzato per il task potrebbe rivelarsi particolarmente efficace.

Il classificatore basato su *autoencoder* migliora leggermente le performance del classificatore semplice, circa del **10% sia in accuracy che in top-5 accuracy** (Tabella 2). Il risultato non è una sorpresa, l'*autoencoder* convoluzionale, anche con a disposizione tanti dati di addestramento grazie alla *data augmentation* e raggiungendo una *loss* molto bassa, non riesce ad ottenere una ricostruzione perfetta delle immagini, come osservabile in Figura 4. Questo vuol dire che anche la rappresentazione del *layer latente* non è così dettagliata da garantire sufficienti informazioni sia per la ricostruzione delle immagini in fase di *decoding* sia, nel caso in analisi, per la classificazione efficace dei fiori rappresentati.

Le performance ottenute sul test set da parte del modello basato sulla *feature extraction* e quello basato sul *fine tuning* sono molto simili. Entrambi i modelli ottengono una **accuracy del 72% e una top-5 accuracy del 90%** (Tabelle 3 e 4). Il **modello basato sulla feature extraction** ottiene però una *loss* leggermente più bassa, di conseguenza viene considerato come il modello migliore tra tutti quelli presentati per risolvere il task. Contrariamente da quanto ci si potrebbe aspettare, il modello basato sul *fine tuning* non ottiene performance superiori rispetto a quello basato sulla *feature extraction*.

Andando ad osservare i grafici di addestramento dei vari modelli ottimizzati osserviamo come in generale il *learning rate* scelto per l'apprendimento sia adatto alla situazione proposta, generando curve di accuracy e *loss* né troppo né troppo poco ripide: **l'apprendimento è graduale** e il *plateaux* è raggiunto dolcemente. Nessun modello ottimizzato è andato oltre le 45 epoch di addestramento, pur avendone a disposizione fino a 100: gli addestramenti sono stati fermati dall'*early stop* imposto in fase di training.

Non sono presenti fenomeni di *overfitting* molto evidenti anche se una lieve tendenza all'*overfitting* sul training set è osservabile in particolare per il modello basato su *autoencoder*: l'accuracy sul training e sul validation set a fine addestramento si distaccano del 35% (Tabella 2 e Figura 10).

Osservando le performance dettagliate del modello migliore (Appendice B) si può notare come la *precision* e la *recall* tra le diverse classi oscillino parecchio da classe a classe, alternando valori molto buoni (alcune classi hanno *precision o recall* anche di 1) ad altri meno validi. In particolare la specie "*Bird of Paradise*" ottiene una **F1-score di 0.977**, la più altra tra tutte le classi, probabilmente per la sua forma e colorazione molto particolare e distinguibile (osservabile nell'appendice A); la specie "*Desert-Rose*" ottiene invece la **F1-score più bassa (0.390)**, principalmente per colpa di una *recall* molto bassa.

Andando ad osservare alcuni degli errori commessi dal modello basato su *feature extraction* in fase di classificazione (Figura 16, 17, 18) è possibile ipotizzare la causa dei principali errori. In Figura 17 è possibile osservare come per il fiore "*Passion Flower*" siano minimi gli errori commessi nel suo riconoscimento, probabilmente per le sue carat-

teristiche che lo rendono particolarmente unico (corona intorno ai petali, colore striato, corolla molto grane). In Figura 16, che mostra gli errori commessi per la classe "*Artichoke*", il carciofo, il modello tende a commettere qualche errore confondendo (anche se solo per 4 casi) con il fiore "*Spear Thistle*", ragionevole date le caratteristiche cromatiche e di forma molto simili. Ancora più evidenti sono gli errori commessi nella classificazione del *fiore di loto* (*Lotus*, Figura 18) che nel 15% dei casi viene confuso con le *ninfie* (*Water Lily*), che come si vede dalle due immagini rappresentative condividono caratteristiche particolarmente simili tra loro, per forma, colori e dimensioni, tanto da renderle difficilmente distinguibili anche per un umano particolarmente attento.

In generale, andando ad osservare il comportamento del modello su altre classi, che per brevità non sono state riportare, è possibile notare una generale difficoltà nel distinguere classi che hanno caratteristiche simili tra loro. Questo comportamento del modello non è una sorpresa, soprattutto alla luce delle performance ottenute, e ci dimostra che l'approccio proposto si può ritenere valido: sarebbe stato molto più grave se il modello confondesse classi molto diverse tra loro. Avendo a disposizione più dati per il training probabilmente le performance del modello proposto, o di altri modelli simili, migliorerebbero, questo perché i modelli apprenderebbero più facilmente le caratteristiche distinctive di ogni specie proposta.

Si confrontano ora performance di **altri modelli** presentati nella **letteratura** e che affrontano la **stessa problematica**. In [1], vengono presentate le performance di un classificatore che utilizza come feature le rappresentazioni SIFT (Scale Invariant Feature Transforma), HOG (Histogram of Oriented Gradients) e HSV (Hue, Saturation, Value) delle immagini; le performance di accuracy ottenute dal classificatore sul dataset di **test segmentato** (con lo sfondo rimosso) sono del **72.8%**. In [6] viene presentato un approccio di feature extraction basato sulla rete **OverFeat** e che utilizza come **classificatore una SVM**, le performance ottenute sul dataset non segmentato sono del 74.7% di accuracy e dell'**86.8%** di accuracy con il dataset non segmentato ma a cui è stata applicata la **data augmentation**. In [7], vengono presentati due modelli convoluzionali, uno basato su AlexNet e uno basato su **GoogleNet** che ottengono, sulla versione segmentata del dataset performance rispettivamente del 43% di accuracy (e 68% di top-5 accuracy) per il modello basato su AlexNet, mentre del **47%** (e **69% di top-5**) per il modello basato su GoogleNet.

## 6 Conclusioni

Per il problema della classificazione di 102 diverse specie di fiori è stato mostrato come, tra 4 prototipi di modelli con approcci diversi, si è dimostrato migliore quello basato sull'**utilizzo della feature extraction** a partire da una rete *VGG16*. Le performance ottenute da quest'ultimo sono sicuramente soddisfacenti, anche se, alla luce dei risultati ottenuti, non è possibile ritenere il problema completamente risolto. Una **accuracy del 72%** è decisamente un buon risultato, vista soprattutto la difficoltà del task, però, allo stesso tempo, non è un risultato particolarmente eccezionale. Anche la misura di **top-5-accuracy** ottiene un valore buono, pari al **90%**, ma anche in questo caso migliorabile. Questo è soprattutto evidente comprando i risultati ottenuti in questo lavoro con quelli presenti in letteratura, l'approccio proposto è efficace, ottendendo risultati migliori di altri approcci (ad esempio quello basato sulle feature SIFT in [1]) ma inferiori rispetto all'utilizzo di altre topologie di reti neurali ([6]).

L’ottimizzazione dei modelli tramite *AutoML* e *Bayesian Optimization* è sicuramente un approccio interessante che si è dimostrato particolarmente efficace nel migliorare le performance sul task. Una procedura *grid-search* per l’ottimizzazione dei modelli sarebbe sicuramente stata più dispendiosa e probabilmente meno efficace di quella effettuata.

È lasciata per successive sperimentazioni l’utilizzo di altri modelli per il *feature extraction* o per il *fine tuning*, così come l’uso di altri ottimizzatori per le reti neurali proposte: entrambi i cambiamenti potrebbero portare a miglioramenti nelle performance finali. Un ulteriore miglioramento delle performance si otterrebbe avendo a disposizione più dati per il training del dataset allo scopo di rendere sempre meno necessario l’utilizzo di tecniche di *data augmentation*.

## Riferimenti bibliografici

- [1] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 2008, pp. 722–729.
- [2] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [3] M. Lindauer, K. Eggensperger, M. Feurer, S. Falkner, A. Biedenkapp, and F. Hutter, “Smac v3: Algorithm configuration in python,” <https://github.com/automl/SMAC3>, 2017.
- [4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.
- [6] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “Cnn features off-the-shelf: an astounding baseline for recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.
- [7] A. Gurnani, V. Mavani, V. Gajjar, and Y. Khandhediya, “Flower categorization using deep convolutional neural networks,” *arXiv preprint arXiv:1708.03763*, 2017.

# Appendici

## A Specie presenti nel dataset

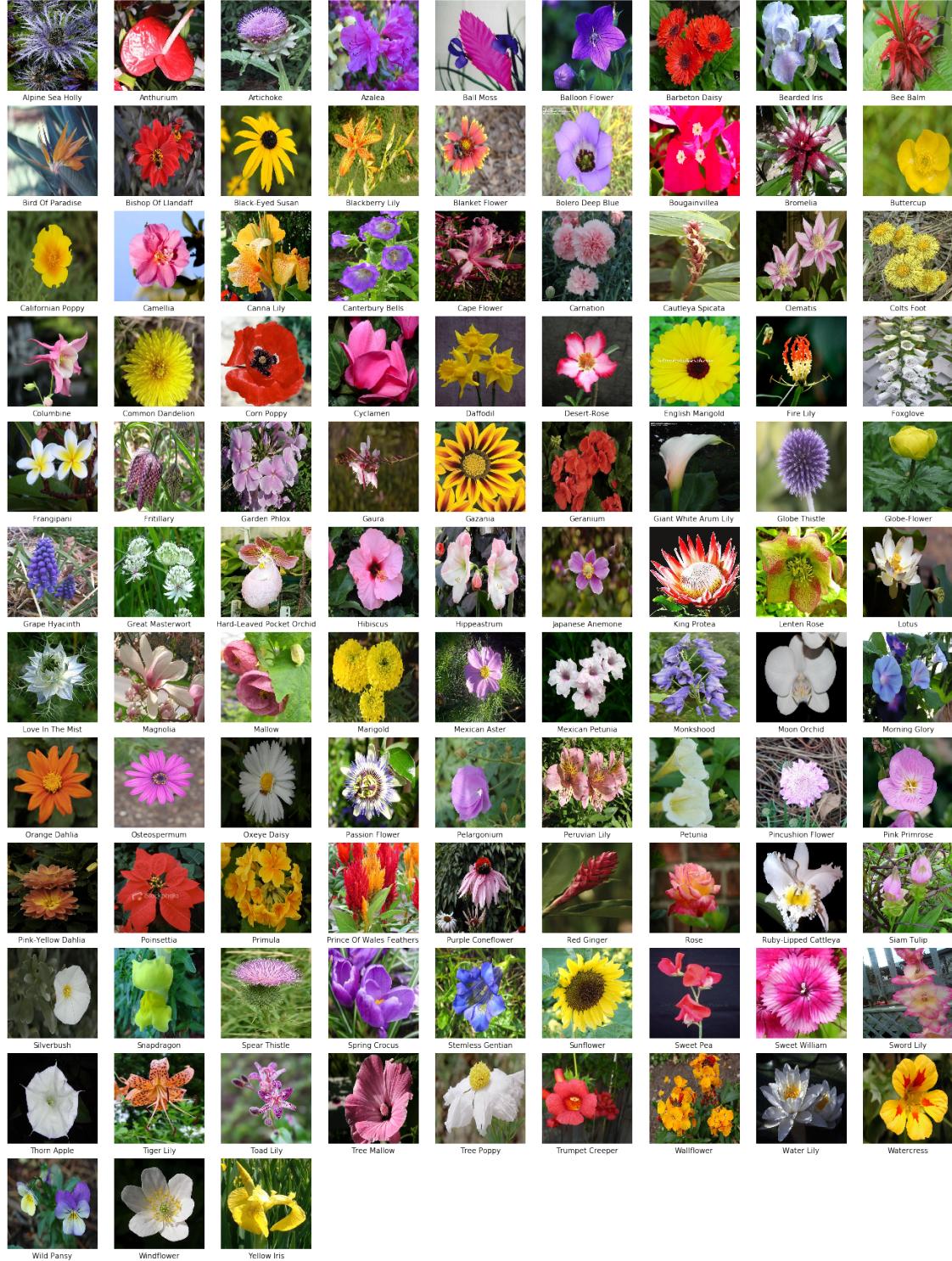


Figura 19: Tutte le specie di fiori presenti nel dataset

## B Performance su test set del modello basato su Feature Extraction

Specie	Precision	Recall	F1	N° Istanze
Alpine Sea Holly	0.826	0.950	0.884	20
Anthurium	0.471	0.909	0.620	44
Artichoke	0.879	0.981	0.927	52
Azalea	0.697	0.351	0.467	151
Ball Moss	0.577	0.750	0.652	20
Balloon Flower	0.828	0.558	0.667	43
Barbeton Daisy	0.748	0.889	0.812	90
Bearded Iris	0.676	0.622	0.648	37
Bee Balm	0.739	0.810	0.773	42
Bird Of Paradise	0.985	0.970	0.977	66
Bishop Of Llandaff	0.809	1.000	0.894	72
Black-Eyed Susan	0.941	0.914	0.928	35
Blackberry Lily	0.893	0.893	0.893	28
Blanket Flower	0.862	0.556	0.676	45
Bolero Deep Blue	0.700	0.519	0.596	27
Bougainvillea	0.444	0.842	0.582	57
Bromelia	0.930	1.000	0.964	40
Buttercup	0.667	0.919	0.773	37
Californian Poppy	0.939	0.647	0.766	119
Camellia	0.310	0.611	0.411	36
Canna Lily	0.806	0.385	0.521	130
Canterbury Bells	0.550	0.151	0.237	73
Cape Flower	0.932	0.943	0.937	87
Carnation	0.656	0.488	0.560	43
Cautleya Spicata	0.967	0.784	0.866	37
Clematis	0.402	0.860	0.548	43
Colts Foot	0.881	0.868	0.874	68
Columbine	0.409	0.474	0.439	57
Common Dandelion	0.903	0.890	0.897	73
Corn Poppy	0.810	0.654	0.723	26
Cyclamen	0.433	0.784	0.558	74
Daffodil	0.538	0.636	0.583	33
Desert-Rose	0.558	0.300	0.390	80
English Marigold	0.578	0.765	0.658	34
Fire Lily	0.900	0.783	0.837	23
Foxglove	0.831	0.967	0.894	122
Frangipani	0.884	0.963	0.921	134
Fritillary	0.845	0.984	0.909	61
Garden Phlox	0.400	0.244	0.303	41
Gaura	0.851	0.678	0.755	59
Gazania	0.741	0.977	0.843	44
Geranium	0.989	0.912	0.949	102
Giant White Arum Lily	0.722	0.839	0.776	31

Specie	Precision	Recall	F1	N° Istanze
Globe Thistle	0.960	0.857	0.906	28
Globe-Flower	0.762	0.571	0.653	28
Grape Hyacinth	0.762	0.842	0.800	19
Great Masterwort	0.833	0.492	0.619	61
Hard-Leaved Pocket Orchid	0.950	0.884	0.916	43
Hibiscus	0.486	0.491	0.489	110
Hippeastrum	0.786	0.564	0.657	78
Japanese Anemone	0.457	0.364	0.405	44
King Protea	0.931	0.871	0.900	31
Lenten Rose	0.766	0.500	0.605	72
Lotus	0.701	0.651	0.675	126
Love In The Mist	0.885	0.821	0.852	28
Magnolia	0.884	0.731	0.800	52
Mallow	0.565	0.338	0.423	77
Marigold	0.979	0.958	0.968	48
Mexican Aster	0.900	0.500	0.643	36
Mexican Petunia	0.484	0.600	0.536	50
Monkshood	0.885	0.500	0.639	46
Moon Orchid	0.850	0.739	0.791	23
Morning Glory	0.609	0.883	0.721	60
Orange Dahlia	0.979	0.780	0.868	59
Osteospermum	1.000	0.891	0.943	46
Oxeye Daisy	0.966	0.848	0.903	33
Passion Flower	0.753	0.994	0.857	175
Pelargonium	0.863	0.518	0.647	85
Peruvian Lily	0.435	0.692	0.535	39
Petunia	0.458	0.784	0.578	139
Pincushion Flower	0.821	0.667	0.736	48
Pink Primrose	0.750	0.556	0.638	27
Pink-Yellow Dahlia	0.955	0.988	0.971	86
Poinsettia	0.973	0.542	0.696	131
Primula	0.548	0.556	0.552	72
Prince Of Wales Feathers	1.000	0.952	0.976	21
Purple Coneflower	0.892	0.879	0.885	66
Red Ginger	0.955	0.778	0.857	27
Rose	0.404	0.884	0.555	69
Ruby-Lipped Cattleya	0.891	0.485	0.628	101
Siam Tulip	0.571	0.261	0.358	46
Silverbush	1.000	0.914	0.955	35
Snapdragon	0.418	0.683	0.519	41
Spear Thistle	0.929	0.765	0.839	34
Spring Crocus	0.727	0.516	0.604	31
Stemless Gentian	0.674	0.912	0.775	34
Sunflower	0.927	0.974	0.950	39
Sweet Pea	0.278	0.667	0.392	15
Sweet William	0.723	0.681	0.701	69

<b>Specie</b>	<b>Precision</b>	<b>Recall</b>	<b>F1</b>	<b>N° Istanze</b>
Sword Lily	0.300	0.917	0.452	36
Thorn Apple	0.900	0.789	0.841	114
Tiger Lily	0.880	0.957	0.917	23
Toad Lily	0.857	0.947	0.900	19
Tree Mallow	0.684	0.565	0.619	46
Tree Poppy	1.000	0.875	0.933	48
Trumpet Creeper	0.763	0.707	0.734	41
Wallflower	0.909	0.751	0.823	213
Water Lily	0.787	0.878	0.830	156
Watercress	0.744	0.878	0.805	139
Wild Pansy	0.892	0.659	0.758	88
Windflower	0.882	0.857	0.870	35
Yellow Iris	0.724	0.778	0.750	27